The Coupon Collector's Problem

The coupon collector's problem is a classic problem in probability theory. It addresses the following question: if you have n different types of coupons and each new coupon you collect is equally likely to be any one of the n types, how many coupons do you need to collect until you have at least one of each type?

The expected number of coupons you need to collect is given by:

$$E(n) = n \left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \right)$$

This sum is approximately equal to $n \log n$ for large n. This is because the harmonic series $\sum_{k=1}^{n} \frac{1}{k}$ is asymptotically equal to $\log n$.

Method 1 Analysis

```
int[] a = new int[n];
boolean[] taken = new boolean[n];
int count = 0;
while (count < n)
{
    int r = StdRandom.uniform(n);
    if (!taken[r])
    {
        a[r] = count;
        taken[r] = true;
        count++;
    }
}
```

- **Initialization**: $O(n)$ for creating arrays.
- **While Loop**: The loop continues until `count < n`.
  - Each iteration involves generating a random number r (constant time, $O(1)$).
  - Checking and updating the `taken` array (constant time, $O(1)$).

In the worst case, each new random number `r` might already be taken, necessitating multiple attempts to find an untaken number. The average number of attempts follows the coupon collector's problem, resulting in $O(n \log n)$ total iterations.

Method 2 Analysis

```
int n = Integer.parseInt(args[0]);
boolean[] isCollected = new boolean[n];
int count = 0;
int distinct = 0;
while (distinct < n)
{
```

```
        int r = (int) (Math.random() * n);
        count++;
        if (!isCollected[r])
        {
            distinct++;
            isCollected[r] = true;
        }
    }
```

- **Initialization**: $O(n)$ for creating the `isCollected` array.
- **While Loop**: The loop continues until `distinct < n`.
  - Each iteration involves generating a random number r (constant time, $O(1)$).
  - Checking and updating the `isCollected` array (constant time, $O(1)$).
  - `count` keeps track of the total number of random numbers generated.

Similar to Method 1, the number of iterations follows the coupon collector's problem. Each new random number might have already been collected, requiring multiple attempts. The average number of attempts to collect all distinct numbers is $O(n \log n)$.

Why $O(n \log n)$?

Both methods essentially simulate the process of collecting coupons:

- In Method 1, `a[r]` and `taken[r]` represent checking and marking an index.
- In Method 2, `isCollected[r]` represents checking and marking an index.

In both cases, as more indices are marked, the probability of randomly hitting an already marked index increases. The number of iterations required to mark all indices grows in proportion to the harmonic series, which approximates $O(n \log n)$.

Therefore, the order of growth for both methods is $O(n \log n)$ because they both are governed by the coupon collector's problem.